Performance Analysis of Multiple Site Resource Provisioning: Effects of the Precision of Availability Information

Marcos Dias de Assunção and Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{marcosd, raj}@csse.unimelb.edu.au

Abstract

Emerging deadline-driven Grid applications require a number of computing resources to be available over a time frame, starting at a specific time in the future. To enable these applications in Grids, it is important to predict the resource availability and utilise this information during provisioning because it affects their performance. It is impractical to request the availability information upon the scheduling of every job due to communication overhead. However, existing work has not focused on how the precision of availability information influences the provisioning. As a result, limitations exist in developing advanced resource provisioning and scheduling mechanisms. In this paper, we investigate how the precision of availability information affects resource provisioning in multiple site environments. Performance evaluation is conducted, considering both multiple scheduling policies in resource providers and multiple provisioning policies in brokers, while varying the precision of availability information. Experimental results show that it is possible to avoid requesting availability information for every Grid job scheduled thus reducing the communication overhead. In addition, they demonstrate that the bounded slowdown of Grid jobs can be improved by policies that use multiple resource partitions.

1 Introduction

Advances in distributed computing have enabled the creation of Grid-based resource sharing networks such as TeraGrid [3, 6], Open Science Grid [1] and PlanetLab [19]. These networks, composed of multiple resource providers, enable collaborative work and resource sharing amongst groups of individuals and organisations. These collaborations, widely known as Virtual Organisations (VOs) [8], require resources from multiple computing sites. The resources contributed by resource providers are generally clusters of computers managed by queueing-based Resource Management Systems (RMSs), such as PBS and Condor.

Emerging deadline-driven Grid applications require access to several resources and predictable Quality of Service (QoS). For example, a given application may require a number of computing resources to be available over a time frame, starting at a specific time in the future. However, it is difficult to provision resources to these applications due to the complexity of providing guarantees about the start or completion times of applications currently in execution or waiting in the queue. Current RMSs use optimisations to the First Come First Served (FCFS) policy such as backfilling to reduce the scheduling queue fragmentation, improve job response time and maximise resource utilisation. These optimisations make it difficult to predict the resource availability over a time frame as the jobs' start and completion times are dependent on resource workloads.

To complicate the scenario further, users may have access to resources via mediators such as brokers or gateways. The design of gateways that provision resources to deadline-driven applications relying on any information given by current RMSs may be complex and prone to scheduling decisions that are far from optimal. Furthermore, it is not clear how gateways can obtain information from providers to provision resources to QoS demanding applications considering current RMSs. Existing work on resource provisioning in Grid environments has used conservative backfilling wherein the fragments of the scheduling queue are given to be provisioned by a broker [25]. These fragments or free time slots correspond to the availability information. We consider impractical to request the free time slots from providers upon the scheduling of every job due to potential communication overhead.

In this paper, we investigate how the precision of availability information affects resource provisioning in multiple site environments. In addition, we enhance traditional schedulers, allowing the obtention of availability information required for resource provisioning. We evaluate the reliability of the provided information under varying conditions by measuring the number of provisioning violations. A violation occurs when the information given by the resource provider turns out to be incorrect when it is used by the gateway. Additionally, we evaluate the impact of provisioning resources to Grid applications on providers' local requests by analysing the job bounded slowdown. We investigate whether EASY backfilling and multiple partition policies provide benefits over conservative backfilling if job backfilling is delayed, enabling large time slots to be provided to the gateway.

The rest of this paper is organised as follows. Section 2 presents the related work. In Section 3 we describe the multiple site resource provisioning scenario and problem. The resource providers' and the gateway's policies are described in Section 4 and Section 5 respectively. We discuss the performance evaluation and experimental results in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

The performance analysis and the policies proposed in this work are related to previous systems and techniques in several manners.

Modelling provider's resource availability: AuYoung *et al.* [2] consider a scenario wherein service providers have contracts with resource providers. The availability information is modelled as ON/OFF intervals whereby ON and OFF intervals correspond to off-peak and peak periods respectively. However, they do not demonstrate in practice how this information can be obtained from resource providers.

Advance reservations and creation of alternative to rejected requests: Approaches for elastic advance reservations and generation of alternative time slots for advance reservation requests have been studied. For example, Röblitz *et al.* present a resource model in which a Grid Reservation Service (GRS) probes clusters to find the possible start times of a job on resources [23]. They propose an algorithm for reserving compute resources from a Grid that allows inexact reservations; the reservations can be elastic in some attributes such as time and number of processors. This kind of advance reservation aims to improve utilisation and give resource providers more flexibility to find appropriate resources, particularly when the reservation of multiple resources for the same time period is necessary. This work has been used as basis to enable co-reservations [22].

Wieczorek *et al.* [26] apply advance reservation to improve the predictability of the execution of time-constraint application workflows on a Grid. The scheduling algorithms take into account sequential jobs. They introduce two algorithms for providers to generate offers to advance reservation requests (i.e. *attentive* and *progressive*). The *attentive* algorithm offers the requested time slots if the resources are available; otherwise it creates alternative offers according to the available slots close to the requested

QoS requirements. *Attentive* tries to keep reserved segments as minimum as possible by providing alternative slots that are adjacent or overlap with existing reservations. The *progressive* algorithm is similar to *attentive* but considers fairness. *Progressive* attempts to distribute the resources of a site fairly amongst the users by preventing them from reserving more resources than allowed. A new phase is introduced to the Heterogeneous Earliest Finish Time (HEFT) algorithm in which the scheduler negotiates with the resource manager in order to reserve one resource for each task of the workflow. Experimental results demonstrate that from low to moderately loaded environments advance reservations improve the performance of workflows.

These models could be incorporated in the provisioning scenario described in this work to improve resource utilisation and generate alternative offers for provisioning violations. However, we aim to reduce the interaction between resource providers and gateways by allowing the providers to inform the gateways about their spare capacity. In this sense, we focus on how the availability information can be obtained from RMSs and how reliable it is under different conditions.

Multiple resource partitions policies: Work on multiple resource partitions and priority scheduling has shown to reduce the job slowdown compared to EASY backfilling policies [15]. We build on this effort and extend it to enable other multiple partition policies. We also propose a new multiple resource partition policy based on load forecasts for resource provisioning.

Resource allocation in consolidated centres: Padala *et al.* [18] apply control theory to address the provision of resources to multi-tier applications in a consolidated data centre. The experimental scenario is composed of two physical hosts, running two virtual machines each. It is shown that the system is able to adjust the resource shares allocated to each virtual machine based on the demands of the applications when the virtual machines share the same CPU. In addition, Garbacki and Naik [10] consider a scenario wherein customised services are deployed on virtual machines which in turn are placed into physical hosts. Services with no isolation requirements can share the same virtual machine as long as the other requirements are met. Although the provisioning of resources to applications in utility data centres is a topic of great importance, in this work we focus on traditional queue-based RMSs and do not consider a consolidated data centre.

Shared spaces for collaborative scheduling: Ranjan *et al.* [20] present a federation of Alchemi desktop clusters wherein Grid Federation Agents (GFAs) responsible for the federated clusters publish information and carry out resource discovery using a Distributed Hash Table (DHT) based P2P network. A shared space (i.e. a peer-to-peer tuple space) is introduced to coordinate the matching of application requirements and resources available at the clusters [21]. Such a shared space can be used to co-ordinate the allocation of resources from providers by gateways. However, we leave the exploration of such an approach for future work in a scenario with multiple gateways exchanging resource shares previously obtained from providers.

Resource provisioning: Singh *et al.* [24, 25] present a provisioning model where Grid sites provide information on the time slots over which sets of resources will be available. The sites provide their resources to the Grid in return for payments, thus they present a cost structure consisting of fixed and variable costs over the resources provided. The provisioning model is evaluated considering the scheduling of workflow applications. The main goal is to find a subset of the aggregated resource availability, termed resource plan, such that both the allocation cost and the application makespan are minimised. They utilise a Multi-Objective Genetic Algorithm (MOGA) approach to approximate the group of resource plans that correspond to the *Pareto-optimal* set. Experiments have been carried out considering one cluster and one broker at time. Our work differs from that by Singh *et al.* in the sense that we investi-

gate multiple approaches to obtain availability information and investigate how reliable this information can be in multiple site environments.

3 Multiple-Site Resource Provisioning

The multiple site scenario we consider is depicted by Figure 1. Although the names of the components derive from our previous work on interlinking Grids [5], the scenario is general enough to reflect the case of a community broker provisioning resources to multiple user applications.

A Resource Provider (RP) contributes a share of computational resources, storage resources, networks, application services or other type of resource to a Grid in return for regular payments. A RP has local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to an InterGrid Gateway (IGG) by providing information about the resources available in the form of free time slots. A free time slot includes information about the number of resources available, their configuration and time frame over which they will be available. The resources provided can be physical or virtual resources such as Virtual Machines (VMs) and the delegation can be made through a secure protocol such as SHARP [9].

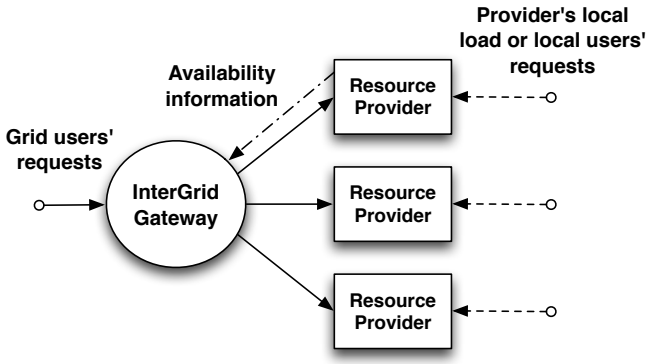

Figure 1. Resource providers contribute to the Grid but have local users.

A Grid can have peering arrangements with other Grids managed by IGGs and through which they coordinate the use of resources. These peering arrangements, however, are not discussed here [5]. In this work, we investigate how an IGG can provision resources to applications based on the availability information given by the resource providers.

Problem Description: IGG_i attempts to provision resources to meet the QoS demanded by its users, improve the job bounded slowdown and minimise the number of violations. A violation occurs when a user tries to use the resources allocated by the IGG and the resources are no longer available due to wrong or imprecise availability information given by the resource provider. Resource providers, on the other hand, want to increase the resource utilisation, thus increasing their profit, without compromising their local users requests. IGG_i should achieve a set of allocations that minimises the response time and bounded slowdown of Grid users' requests without perceivable impact on the slowdown of the RPs' local requests.

Grid Requests: A request is contiguous and needs to be served with resources from a single resource provider. The requests received by an IGG contain a description of the required resources and the request duration. A request can either demand QoS or require a best effort service. A QoS constrained request has an execution estimate, a deadline and a ready time before which the request is not available for scheduling. A best effort job has an execution time estimate but does not have a deadline.

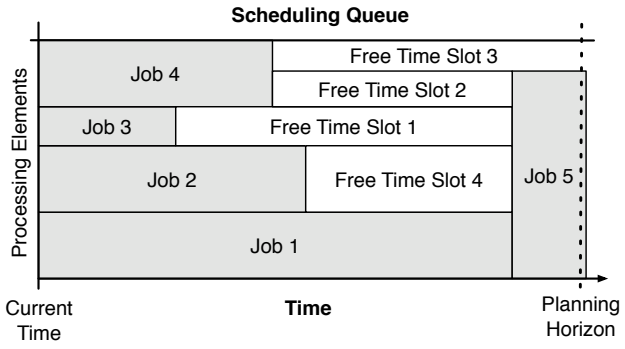

Figure 2. Free time slots (conservative backfilling).

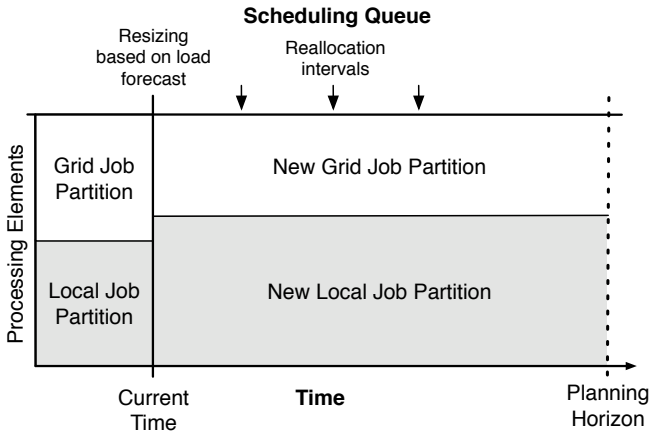

Figure 3. Free time slots (multiple partitions).

4 Resource Provider Policies

We have extended traditional scheduling policies in order to obtain the free time slots from resource providers. The policies described here utilise an 'availability profile' similar to that described by Mu'alem and Feitelson [17]. The availability profile is a list whose entries describe the CPUs available at particular times in the future. These correspond to the completion or start times of jobs and advance reservations. Jobs with the same completion time or scheduled to start at the completion of another job share entries in the profile.[1] By scanning the availability profile and using other techniques described here, the resource providers inform the gateway about the free time slots; the gateway in turn can carry out provisioning decisions based on this information.

Conservative Backfilling Based Policies: Under conservative backfilling, a job is used to backfill and start execution earlier than expected, given that it does not delay any other job in the scheduling queue [17]. In order to reduce complexity, the schedule for the job is generally determined at its arrival and the availability profile is updated accordingly. Given those conditions, it is possible to obtain the free time slots by scanning the availability profile. This approach, depicted in Figure 2, was also used by Singh *et. al* [24, 25] to extend a conservative backfilling based RMS. In that case, the availability profile is scanned until a given time horizon thus creating windows of availability or free time slots; the finish time of a free time slot is either the finish time of a job in the waiting queue or the planning horizon. We have also implemented a conservative backfilling policy that uses multiple resource partitions based on the EASY backfilling proposed by Lawson and Smirni [15].

Multiple Resource Partition Policies: We have implemented other policies based on multiple resource partitions. In our implementation, the policy divides the resources available in multiple partitions and assigns jobs to these partitions according to partition predicates. A partition can borrow resources from another when they are not in use by the latter and are allowed by the scheduler. One of the policies is the implementation of the EASY backfilling (aka aggressive backfilling) described by Lawson and Smirni [15]. In this case, each partition uses aggressive backfilling and has a pivot, which is the first job for the partition waiting in the queue. A job belonging to a given partition can start its execution given that it does not delay the partition's pivot and the partition has enough resources. If the partition does not have enough resources, the job can still start execution if additional resources can be borrowed from

[1] An advance reservation may require two entries in the list to mark its start and finish times respectively.

other partitions without delaying their pivots. Additionally, Lawson and Smirni use priority scheduling wherein the waiting queue is ordered by priority when the scheduler is backfilling. A high priority job can replace a lower priority pivot in its partition. In order to evaluate this policy, we attempt to maintain the configuration provided by Lawson and Smirni [15], which selects partitions according to the jobs' runtimes. The partition p for a job is selected according to Equation 1, where t_r is the job's runtime in seconds.

$$p = \begin{cases} 1, & 0 < t_r < 1000 \\ 2, & 1000 \leqslant t_r < 10000 \\ 3, & 10000 \leqslant t_r \end{cases} \quad (1)$$

We also introduce a new policy, depicted in Figure 3, in which, at time intervals, the partitions are resized by the scheduler based on a load forecast computed from information collected at previous intervals. As load forecasts are prone to be imprecise, when the scheduler resizes partitions, it also schedules reallocation events. At a reallocation event, the scheduler evaluates whether the load forecast has turned out to be an underestimation or not. If the forecast load was underestimated, the policy resizes the partitions according to the load from the last resizing period until the current time and backfill the jobs, starting with the local jobs.

Algorithm 1 describes in more detail two procedures used by the load forecast policy; *getFreeTimeSlots* is invoked every time the provider needs to send the availability information to the gateway whereas *reallocationEvent* is triggered by *getFreeTimeSlots* to verify whether the previous forecast has turned out to be precise or whether a reallocation is required.

We use EASY backfilling with configurable maximum number of pivots, similarly to MAUI scheduler [14]. This enables the policy to be converted to conservative backfilling by setting the maximum number of pivots to a large value, here represented by ∞. From line 3 to 4 of Algorithm 1 the scheduler becomes conservative backfilling based by setting the number of pivots in each partition to ∞. It also schedules the jobs currently waiting in the queue. After that, the scheduler returns to EASY backfilling (line 5). Then, from line 6 to 10, the scheduler obtains the load forecast and the free time slots and resizes the free time slots by modifying the number of CPUs according to the amount of resources expected to be available over the next interval. Next, the scheduler triggers a reallocation event. From line 20 to 24 the scheduler verifies whether the forecast was underestimated. If that is the case, it throws the towel and turns the policy to conservative backfilling and informs the gateway about the availability.

5 Gateway Provisioning Policies

The policies we consider for the gateway are described as follows:

- **Least loaded resource:** The gateway submits a job to the least loaded resource based on utilisation information sent by the resource providers every ten minutes.
- **Earliest start time:** This policy is employed for best effort and deadline constrained requests when the resource providers are able to inform the gateway about the free time slots.

Algorithm 2 illustrates the earliest start time schedule performed by the gateway. When scheduling a job, the algorithm is given the provider's availability information and the job. If the providers send the information at regular time intervals, this information is already available at the gateway; otherwise, the gateway requests it from the resource providers. If the job is not deadline constrained, the gateway

```
1  procedure getFreeTimeSlots()
2  begin
3      set number of pivots of local and Grid partitions to ∞
4      schedule / backfill jobs in the waiting queue
5      set number of pivots of local and Grid partitions to 1
6      actualLoad ← load of waiting/running jobs
7      forecast ← get the load forecast
8      percToProvide ← min{0, 1 − actualLoad}
9      slots ← obtain the free time slots
10     slots ← resize slots according to percToProvide
11     if percToProvide > 0 then
12         inform gateway about slots
13         schedule reallocation event
14     schedule next event to obtain free time slots
15 end
16 procedure reallocationEvent()
17 begin
18     localLoad ← obtain the local load
19     forecast ← get the previously computed forecast
20     if localLoad > forecast then
21         set number of pivots of local partition to ∞
22         schedule / backfill jobs in the waiting queue
23         set number of pivots of grid partition to ∞
24         schedule / backfill jobs in the waiting queue
25         slots ← obtain the free time slots
26         inform gateway about slots
27     else
28         schedule next reallocation event
29 end
```

Algorithm 1: Provider's load forecasting policy.

selects the first provider and submits the job to it. When the job is deadline constrained, the gateway attempts to make a reservation for it. If the reservation cannot be accepted by the provider, the provider updates its availability information (referred in the algorithm as *options*). The gateway then updates the availability information considering the new scheduled job.

Storing Free Time Slots at the IGG: The resource providers issue free time slots and send them to the IGG on a periodical basis or at request. The IGG maintains the availability information given by a provider on a modified red-black tree [4]. Each node has two references namely to its predecessor and successor nodes thus forming a linked list. This tree is analogous to the availability profile described by Mu'alem and Feitelson [17]; the nodes are ordered according to their times. That is, a free slot may lead to the creation of two nodes in the tree, namely to mark its start and finish times; free time slots can share nodes.

6 Performance Evaluation

6.1 Scenario Description

We have modelled DAS-2 Grid configuration. DAS-2 has been selected because job traces collected from this Grid and the resource configuration are publicly available and have been previously stud-

```
input: list of providers providers and job job
num_prov ← providers.size
trial ← 1
estimates ← ∅
foreach provider ∈ providers do
    slot ← get earliest time slot from provider's info
    estimates ← estimates ∪ slot
repeat
    sort estimates by order of start time
    slot ← first element of estimates
    provider ← slot.provider
    cpus ← job.num_cpus
    if job is not deadline constratined then
        submit(job, provider)
        break repeat
    else
        duration ← job.runtime
        start ← slot.start_time
        reservation ← reserve(cpus, start, duration, provider)
        if reservation is successful then
            submit(job, provider, reservation)
            break repeat
        else
            options ← reservation.options
            update provider's avail. info with options
            estimates ← estimates ∩ slot
            slot ← get earliest time slot from options
            estimates ← estimates ∪ slot
    trial ← trial + 1
until trial = num_prov
```

Algorithm 2: Gateway's scheduling procedure.

ied [12]. As depicted in Figure 4, DAS-2 is a Grid infrastructure deployed in the Netherlands comprising 5 clusters. The evaluation of the proposed mechanism is performed through simulation by using a modified version of GridSim.[2] We resort to simulation as it provides a controllable environment and enables us to carry out repeatable experiments.

We model the resource providers' local jobs according to the workload model proposed by Lublin and Feitelson [16]; we refer to this model as Lublin 99. We configure the Lublin 99 model to generate typeless jobs (i.e. we do make distinctions between batch and interactive jobs); the maximum number of CPUs used by the generated jobs is set accordingly to the number of nodes in the clusters; we generate four month long workloads. The characteristics of Grid jobs, such as arrival rate, number of processors required and execution time are modelled using DAS-2 job trace available at the Grid Workloads Archive.[3]. We use the interval from the 9th to the 12th month. The jobs' runtimes are taken as runtime estimates. Although this generally does not reflect the reality, it has been shown that it provides the basis

[2]More information about the extensions made to the simulator and the implementation of the policies described is available at http://www.cs.mu.oz.au/~marcosd/software.html

[3]Grid Workloads Archive website: http://gwa.ewi.tudelft.nl/pmwiki/

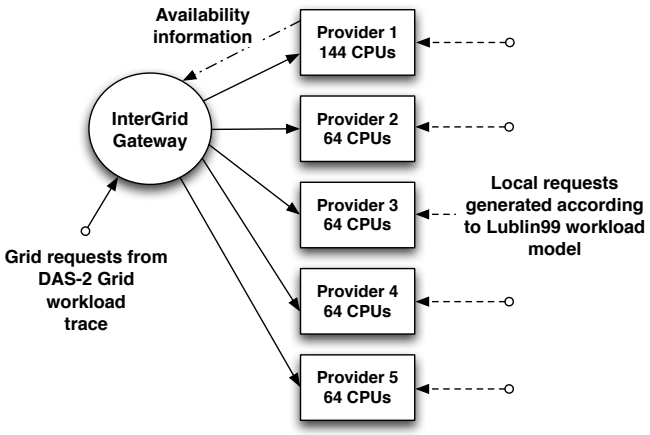

Figure 4. Environment modelled.

or bounds for comparison of scheduling approaches [7].

We make an attempt to simulate the steady-state of the system. This way, the last simulated event is the arrival of the last job submitted in any of the workloads. Additionally, we attempt to eliminate the system warm-up by disregarding the first two weeks of the experiments. In the case of the forecast based policy, the second week is used as training period. As described beforehand, there are two types of requests, namely deadline constrained and best-effort. We select randomly the requests that will be deadline constrained. In order to generate the request deadlines we use a technique described by Islam *et al.* [13], which provides a feasible schedule for the jobs. We perform the experiments by using the same Grid environment using aggressive backfilling at the resource providers and 'submit to the least loaded resource' policy at the gateway. A request deadline is the job completion under this scenario multiplied by a *stringency factor*. For load forecasting we use a weighted exponential moving average [11], considering a window of 25 intervals.

Performance Metrics: One of the metrics considered is the bounded job slowdown ($bound$=10 seconds) hereafter referred to as job slowdown for short [7]. Specifically, we measure the bounded slowdown improvement ratio $\mathcal{R}$ given by Equation 2, where s_{base} is the job slowdown using a base policy used for comparison; and s_{new} is the job slowdown given by the policy being evaluated. We calculate the ratio $\mathcal{R}$ for each job and then take the average. The graphs presented in this section show average ratios.

$$\mathcal{R} = \frac{s_{base} - s_{new}}{min(s_{base}, s_{new})} \tag{2}$$

We also measure the number of violations and messages exchanged between providers and gateway to schedule Grid jobs. The reduction in the number of messages required is used for estimating the trade-off between precision of information and communication overhead. A violation indicates whether the information given by a resource provider turned out to be incorrect due to imprecise load estimation or 'bursty' job arrival for example. This helps us to evaluate whether it is possible to perform provisioning based on the information provided by traditional schedulers. A given job j faces a violation when the inequality in Equation 3 is *true*, where j_{gst} is the job start time assigned by the gateway based on the free time slots given by providers; j_{pst} is the actual job start time set by the provider's scheduler; and T is a tolerance time. The experiments performed in this work use a T of 20 seconds. A violation also occurs when a resource provider cannot accept a reservation request made by the gateway.

Table 1. Parameter description.

Parameter	Value
Planning horizon duration	∞
Deadline constrained requests	20%
Stringency factor	5

$$j_{pst} - j_{gst} > T \tag{3}$$

Policy Acronyms: Due to space limitations, we abbreviate the name of the evaluated policies in the following manner. A policy name comprises two parts separated by +. The first part represents the policy employed by the provider whereas the second represents the gateway policy. In the resource provider's side, **Ar** stands for Advance reservation, **Eb** for EASY backfilling, **Cb** for Conservative backfilling, **M** for Multiple partitions and **Mf** for Multiple partitions with load forecast. On the other side, for the gateway's policy, **least-load** means 'submit to least loaded resource', **earliest** represents 'select the earliest start time', **partial** indicates that providers send free time slot information to the gateway on a periodical basis and **ask** means that the gateway requests the free time slot information before scheduling a job. This way, **ArEbMf+earliest-partial** for example, indicates that providers support advance reservation, EASY backfilling, multiple partitions and load forecasts, whereas the gateway submits jobs selecting the earliest start time based on the availability information sent by providers at regular intervals.

6.2 Experimental Results

The first experiment measures the number of messages required by variants of the policies supporting advance reservation and conservative backfilling (i.e. ArCb) that request the free time slots and those in which the time slots are informed by providers at time intervals. We want to investigate whether we can reduce the number of messages required by making the resource providers publish the availability information at gateways at time intervals. In addition, we consider impractical to request the free time slots from providers upon the scheduling of every job due to communication overhead. We vary the interval for providing the availability information; we also measure the number of violations and average job slowdown in each scenario to check the tradeoff between the precision of scheduling decisions and the freshness of the information. We set the planning horizon to ∞, which means that every time a provider sends its availability information to the gateway, it is providing all the free time slots available. In addition, we consider a two phase commit protocol for advance reservations. The time interval for providing the time slots to the gateway is described in the last part of the name of the policies (e.g. 15 min., 30 min.). Around 20% of the Grid requests are deadline constrained. The parameters are summarised in Table 1.

As presented in Figure 5, the number of messages required by the policy in which the gateway asks for the time slots upon the schedule of every job (i.e. ArCb+earliest-ask) creates large number of messages compared to other policies. In contrast, policies that provide the free time slots at regular intervals or when an advance reservation request fails leads to a lower number of messages.

The number of violations increases as the providers send the availability information at larger intervals as depicted by Figure 6. If the scheduling is made based on the free time slots provided every 15 minutes, the number of violations is 973, which accounts for 0.43% of the jobs scheduled. To evaluate whether these violations have an impact on the resource provisioning for Grid jobs, we measure the average bounded slowdown of Grid jobs (Figure 7).

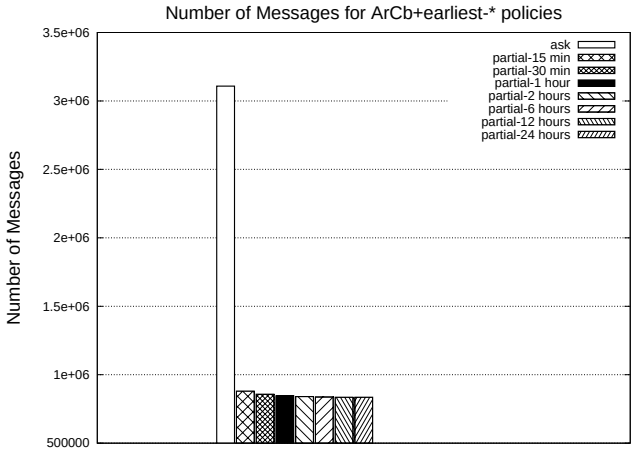

Figure 5. Number of messages by ArCb+earliest-* policies.

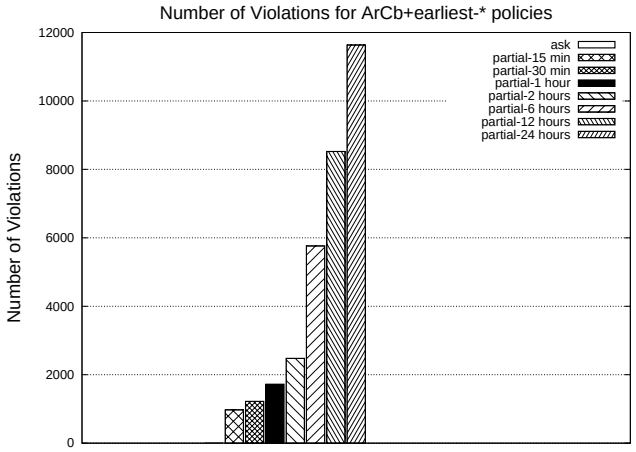

Figure 6. Number of violations by ArCb+earliest-* policies.

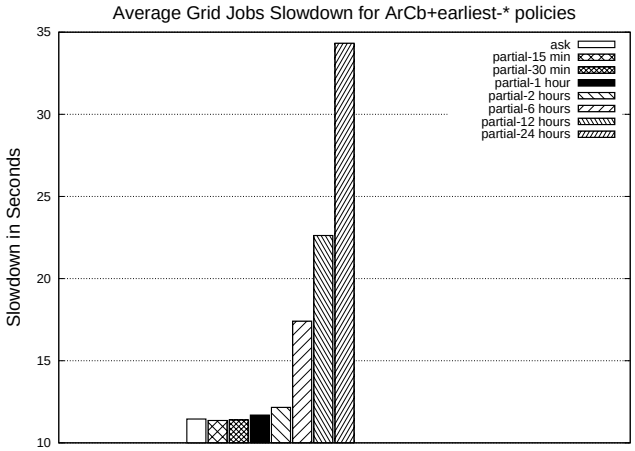

Figure 7. Average job slowdown by ArCb+earliest-* policies.

Figure 7 presents the average slowdown of Grid jobs. As shown, there is an increase in the job slowdown as the interval for providing the free time slots increases. However, when the providers send the availability information every 15 minutes, the average slowdown is improved. From the experiments we can conclude that for a Grid like DAS-2 wherein providers send the availability information at intervals of 15 to 30 minutes resource provisioning can be possible using a simple policy supporting conservative backfilling.

The second experiment we performed follows the approach described by Lawson and Smirni [15] used to evaluate their multiple-queue policy. The values presented in the graphs are averages of 5 simulation rounds each with different workloads for providers' local jobs. We measure average of jobs ratio $\mathcal{R}$ described in Equation 2. Here, the set of policies used as basis for comparison consists in EASY backfilling in the providers and 'submit to the least loaded resource' in the gateway. This way, the experiment measures the average improvement ratio wherein the base policies are aggressive backfilling and submit to the least loaded resource. The resource providers send the availability information to the gateway every two hours. In this experiment we do not consider deadline constrained requests, as they could lead to job rejections by some policies, which would then impact on the average bounded

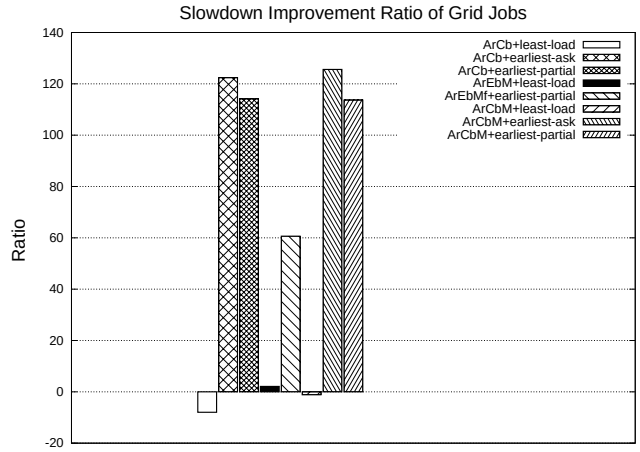

Figure 8. Slowdown ratio of Grid jobs.

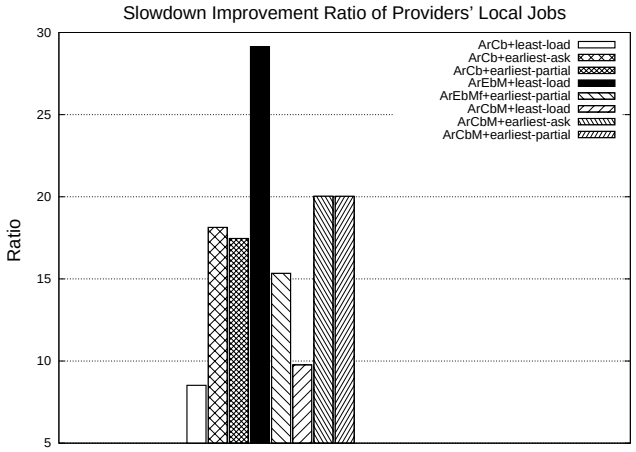

Figure 9. Slowdown ratio of providers' local jobs.

slowdown.

The results show that policies relying on conservative backfilling and 'submit to the least loaded resource' (i.e. ArCb+least-load and ArCbM+least-load) tend to degrade the bounded slowdown of Grid jobs (Figure 8). The reason is that submitting a job to the least loaded resource, wherein utilisation is computed by checking how many CPUs are in use at the current time, does not ensure immediate start of a job because other jobs in the waiting queue may have been already scheduled. Moreover, the gateway is not aware of the time slot the job will in fact utilise.

The multiple resource partition policies utilising conservative backfilling without priorities and providing the free time slots to the gateway improve both the average slowdown of both Grid jobs (Figure 8) and providers' local jobs (Figure 9).

When comparing the EASY backfilling approaches, the policy proposed by Lawson and Smirni [15] (i.e. ArEbM+least-load) improves the slowdown of local jobs (Figure 9) providing little changes in the slowdown of Grid jobs. That occurs because according to the original implementation of this policy, higher priority is given to local jobs. The policy that resizes the resource partitions according to load estimates improves the slowdown of both Grid jobs (Figure 8) and providers' local jobs but not as much as that of the other multiple partition policies.

In order to evaluate the impact of the intervals for providing the availability information, we perform the previous experiment with different time intervals. The results in Figure 10 show that for small planning horizons, the multiple resource partition policy that uses EASY backfilling and load estimates (i.e. ArEbMf+earliest-partial) improves the average ratio, but not as much as the other policies. However, as the time interval for providing the availability information increases, the policy outperforms the other multiple partition policies. The slowdown is improved comparing to the other policies when the intervals for providing the availability information increases. The reason for the better performance under long intervals may be because if a load estimate is wrong, the policy becomes a multiple partition conservative backfilling. When an incorrect estimate is identified in a long interval, it may take a while to approach the next interval when the policy will become EASY backfilling again. This conservative backfillingseems to provide a better slowdown. In addition, updating the availability in the middle of an interval due to a wrong estimate provides an advantage over the other policies. Furthemore, we expected that better load forecast methods could improve the jobs slowdown under varying intervals.

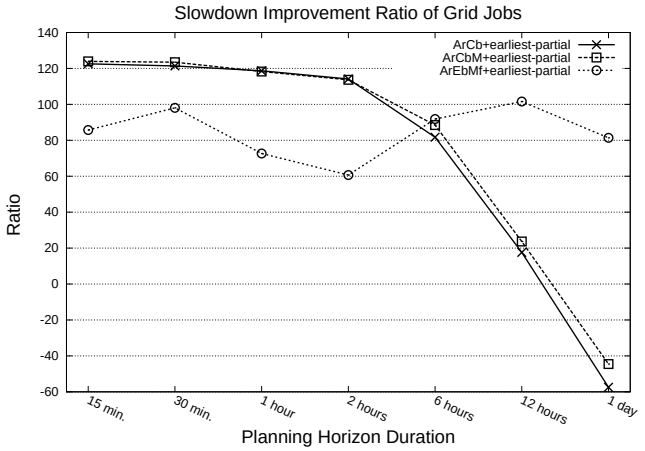

Figure 10. Grid jobs ratio under different horizons.

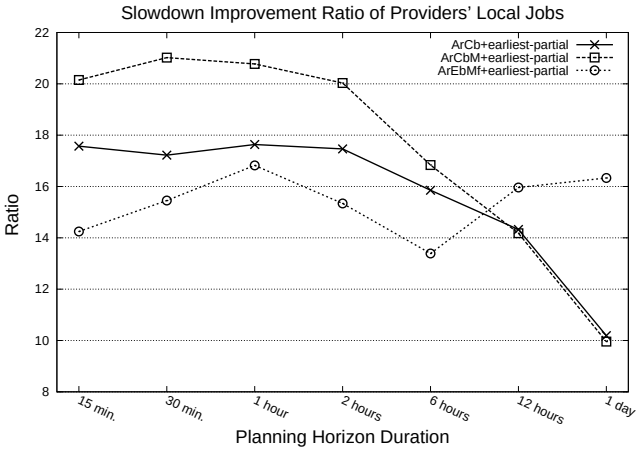

Figure 11. Local jobs ratio under different horizons.

7. Conclusions and Future Work

This work investigates resource provisioning in multiple site environments. It evaluates whether it is possible to provision resources for Grid applications based on availability information given by resource providers using traditional resource management systems. We present empirical results that demonstrate that in an environment like DAS-2, a gateway can provision resources to Grid application if the resource providers inform the available time slots between 15 and 30 minutes. Additionally, multiple resource partition policies can improve the slowdown of both local and Grid jobs if conservative backfilling is used.

We intend to work on provisioning of resources in multi-level federated environments. Future investigations include more sophisticated resource provisioning policies for the gateways, specially for handling advance reservation requests and more sophisticated load forecasting techniques. In addition, we are currently working on request redirection across gateways (i.e. we are working on mechanisms to support transitive relationships between Grids via a contract network between their gateways).

Acknowledgements

We thank Marco Netto, Sungjin Choi, Mukaddim Pathan and Cynthia Fantoni from the University of Melbourne for sharing their thoughts on the topic. We are grateful to the Grid Workloads Archive group for making the Grid workload traces available. This work is supported by DEST and ARC Project grants. Marcos' PhD research is partially supported by National ICT Australia (NICTA).

References

[1] Open Science Grid. http://www.opensciencegrid.org, 2005.

[2] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006)*, pages 119–131, Paris, France, July 2006.

[3] C. Catlett, P. Beckman, D. Skow, and I. Foster. Creating and operating national-scale cyberinfrastructure services. *Cyberinfrastructure Technology Watch Quarterly*, 2(2):2–10, May 2006.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Massachusetts, 2nd edition, 2001.

[5] M. D. de Assunção, R. Buyya, and S. Venugopal. InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience (CCPE)*, 20(8):997–1024, June 2008.

[6] T. Dunning and R. Nandkumar. International cyberinfrastructure: Activities around the globe. *Cyberinfrastructure Technology Watch Quarterly*, 2(1), February 2006.

[7] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing (IPPS '97)*, pages 1–34, London, UK, 1997. Springer-Verlag.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[9] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An architecture for secure resource peering. In *19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 133–148, New York, NY, USA, 2003.

[10] P. Garbacki and V. K. Naik. Efficient resource virtualization and sharing strategies for heterogeneous Grid environments. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 40–49, Munich, Germany, May 2007.

[11] J. E. Hanke and A. G. Reitsch. *Business Forecasting*. Prentice-Hall, Inc., Englewood Cliffs, USA, 5th edition, 1995.

[12] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating Grids through delegated matchmaking. In *2007 ACM/IEEE Conference on Supercomputing (SC 2007)*, Reno, USA, November 2007.

[13] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for parallel job scheduling. In *9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, volume 2862 of *LNCS*, pages 252–268, Seattle, WA, USA, 2003. Springer.

[14] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the Maui scheduler. In *7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '01)*, LNCS, pages 87–102, London, UK, 2001. Springer-Verlag.

[15] B. G. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, LNCS, pages 72–87, London, UK, 2002. Springer-Verlag.

[16] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[17] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[18] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *The 2007 Conference on EuroSys (EuroSys 2007)*, pages 289–302, Lisbon, Portugal, March 2007. ACM Press.

[19] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. PlanetLab architecture: An overview. Technical Report PDN-06-031, PlanetLab Consortium, Princeton, USA, May 2006.

[20] R. Ranjan, X. Chu, C. A. Queiroz, A. Harwood, and R. Buyya. A self-organising federation of Alchemi desktop Grids. Technical Report GRIDS-TR-2007-15, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia, August 2007.

[21] R. Ranjan, A. Harwood, and R. Buyya. Peer-to-peer tuple space: A novel protocol for coordinated resource provisioning. Technical Report GRIDS-TR-2007-14, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia, August 2007.

[22] T. Röblitz and A. Reinefeld. Co-reservation with the concept of virtual resources. In *Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2005)*, volume 1, pages 398–406, Washington, DC, USA, 2005. IEEE Computer Society.

[23] T. Röblitz, F. Schintke, and J. Wendler. Elastic Grid reservations with user-defined optimization policies. In *Workshop on Adaptive Grid Middleware (AGridM 2004)*, Antibes Juan-les-Pins, France, September 2004.

[24] G. Singh, C. Kesselman, and E. Deelman. Application-level resource provisioning on the grid. In *2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006)*, pages 83–83, Amsterdam, The Netherlands, December 2006.

[25] G. Singh, C. Kesselman, and E. Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in Grids. In *16th International Symposium on High Performance Distributed Computing (HPDC 2007)*, pages 117–126, Monterey, California, USA, June 2007. ACM Press.

[26] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan, and T. Fahringer. Applying advance reservation to increase predictability of workflow execution on the Grid. In *Second IEEE International Conference on e-Science and Grid Computing (E-Science 2006)*, page 82, Washington, DC, USA, December 2006. IEEE Computer Society.